

EOS  
CHEMISTRY  
Platform  
GSFC / TRW



# Science Data Processing Framework using UML and Rational Rose

Akbar Thobhani  
Software Engineer  
Jet Propulsion Laboratory  
California Institute of Technology



Final

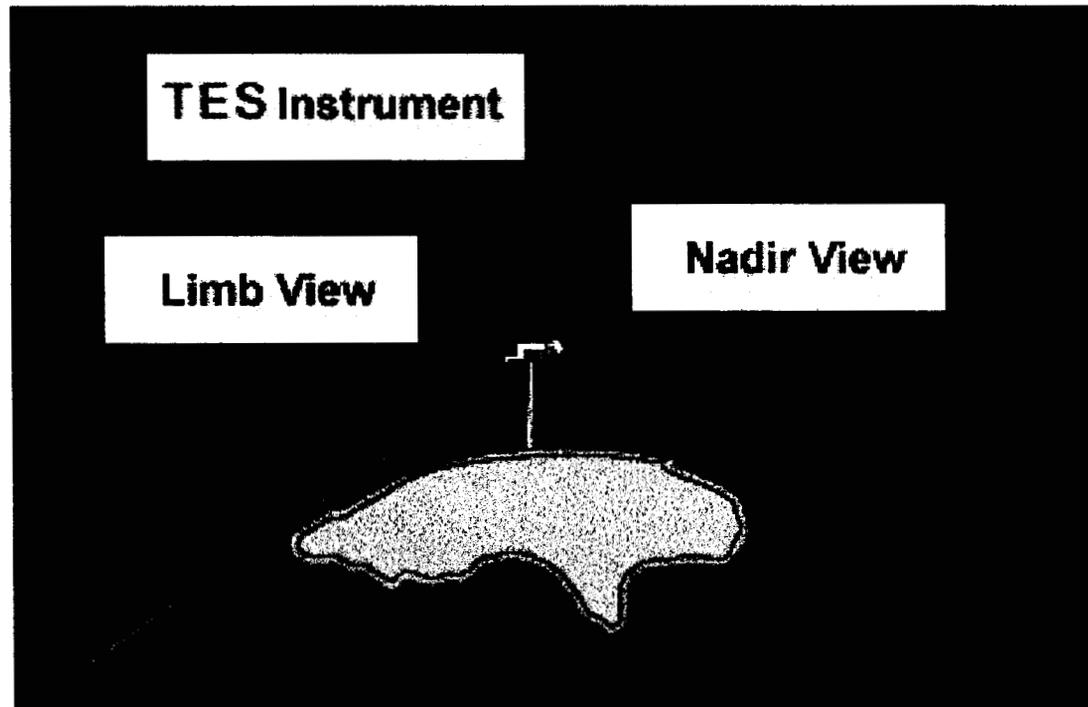
# Agenda

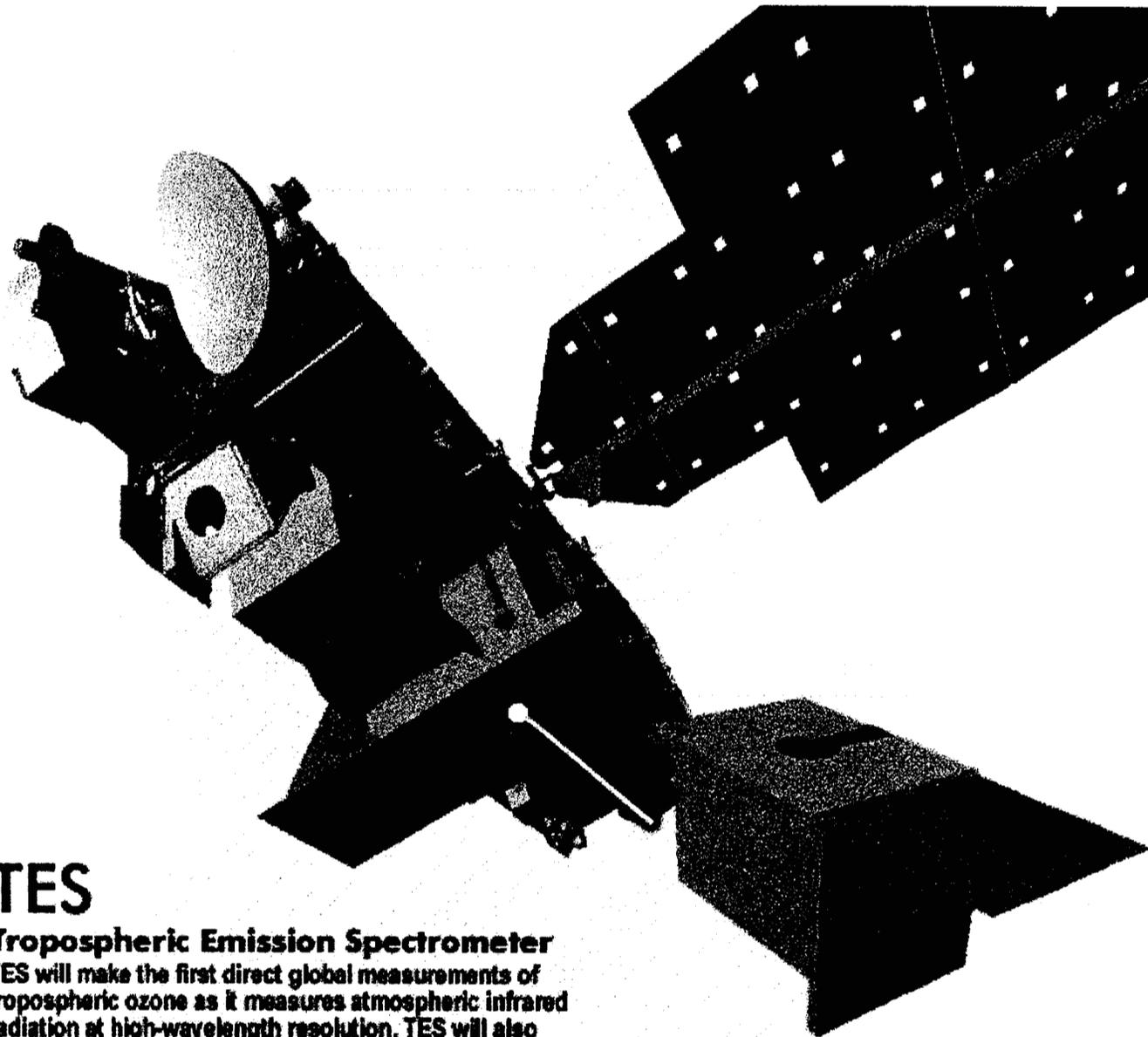
---

- Introduction to TES Framework
- UML
- Rational Rose
- Framework Design
- Lessons Learned
- Questions

# NASA / JPL's TES

---





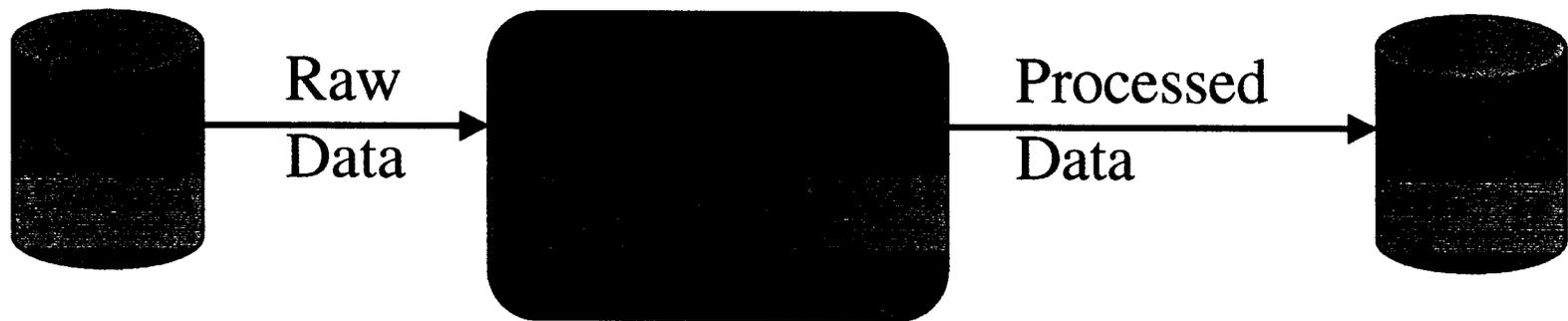
## TES

### **Tropospheric Emission Spectrometer**

TES will make the first direct global measurements of tropospheric ozone as it measures atmospheric infrared radiation at high-wavelength resolution. TES will also measure tropospheric carbon monoxide, methane, nitric acid, water vapor, nitric oxide, and nitrogen dioxide.

# Tropospheric Emission Spectrometer (TES)

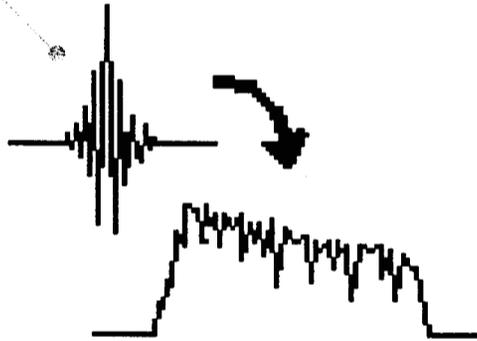
- Provide world's first three-dimensional map of Tropospheric ozone and its photochemical precursors
- Launches in 2002 for a six-year mission



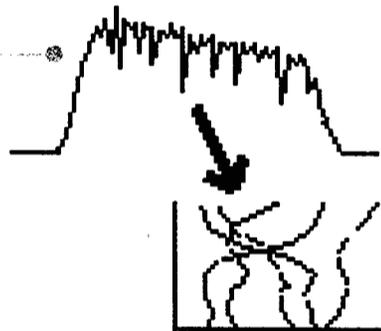
Data processing Per Year

# Data Products Generated

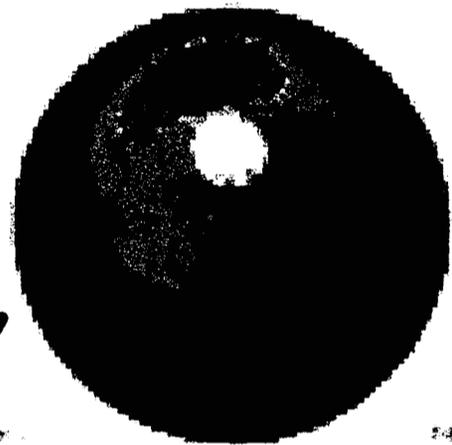
Interferogram



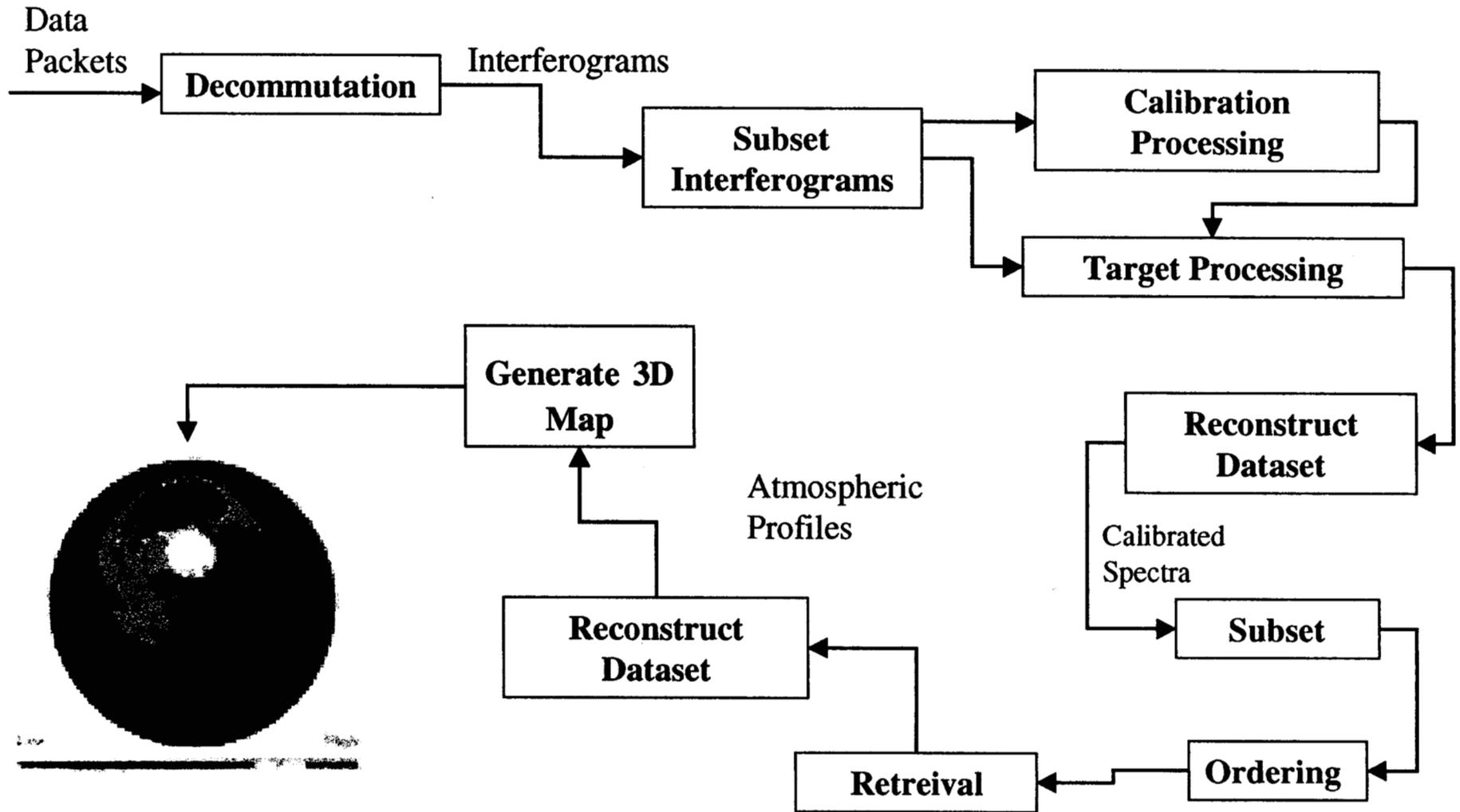
Spectra



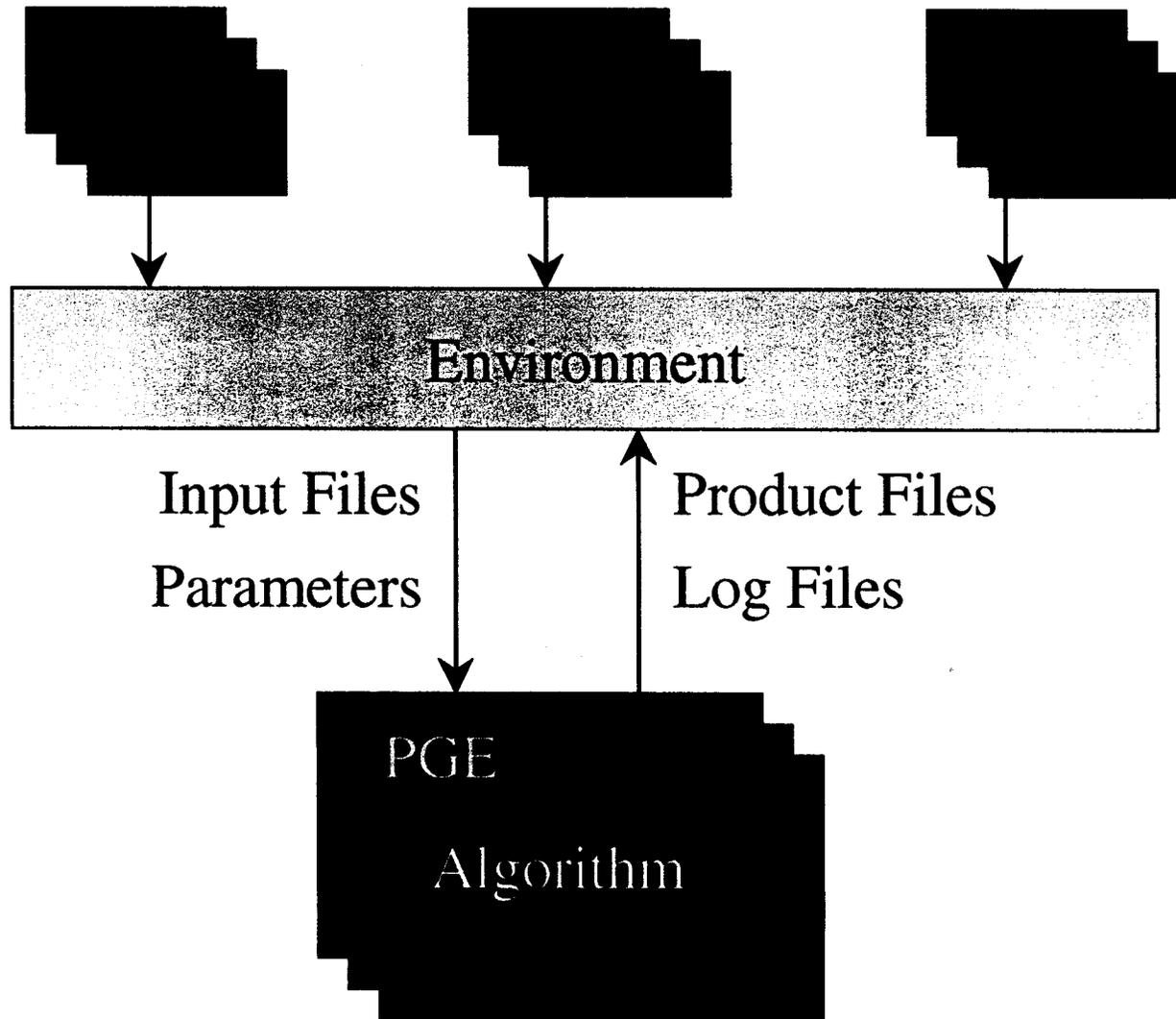
Profiles



# Organization of TES - Product Generation Executable (PGE)



# Data Processing Environment



# Science Data Processing System

---

- Must operate on more than one facility
  - Batch environment
  - Interactive environment
- Provide distributed processing
- Data I/O in more than one format
- Adapt to constant revisions of processing algorithm

# Need for Framework

---

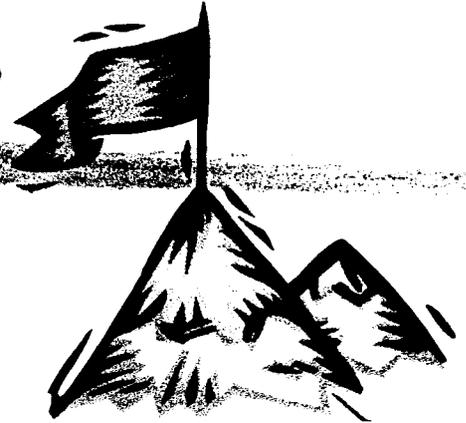
- Abstract PGE related issues and solve them once
- Reduce cost by reusing code
- Provide rapid development of PGEs



*"A partially completed software application that is intended to be customized to completion" - Gregory Rogers*

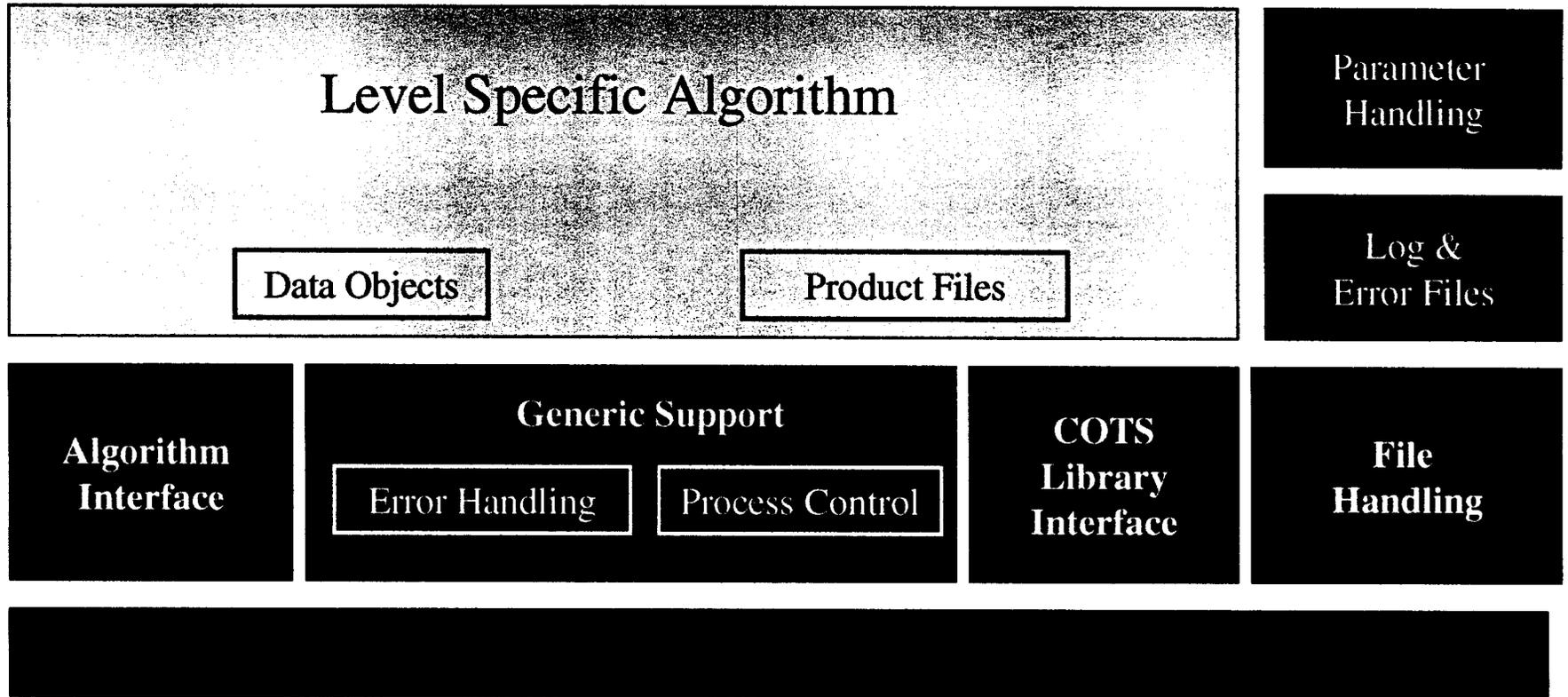
*Framework-based Software Development in C++, Englewood Cliffs, NJ: Prentice-Hall, 1997*

# Framework Design Goals



- Protect science algorithm from environment changes
- Provide PGEs with a generic “shell” for Algorithms
- Implement common functionality (e.g. File I/O, Parameter handling)
- Encapsulate COTS and GOTS Libraries

# SDPS Framework



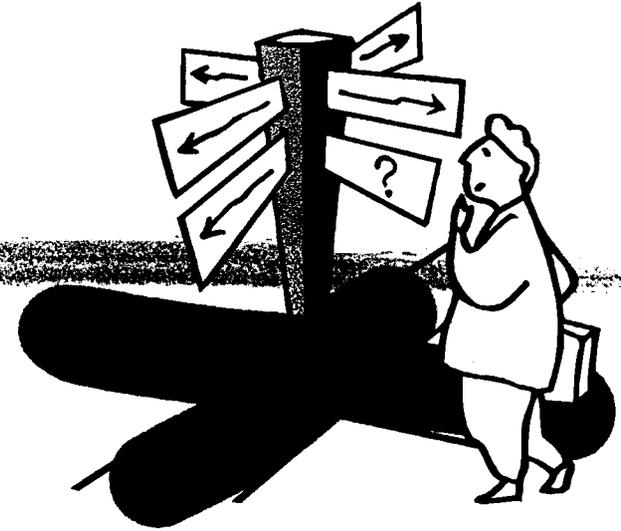
# Using UML

---

**Diagrams we most  
commonly use**



# Analysis



## Data Flow Diagram

- To understand the problem
- To communicate with non OO members of the team

## ■ Use Case Diagrams

- To show coverage of requirements within developers

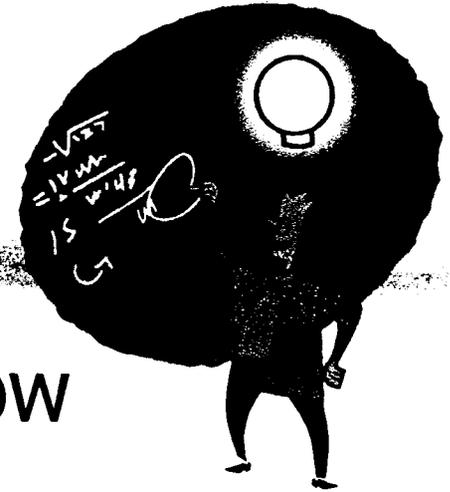
# Conceptual Design

---



- Object Diagram used to show “high level” objects in the system
- Collaboration Diagrams used to show the interaction between these objects.
- Activity Diagram used more like a flow chart

# High Level Design



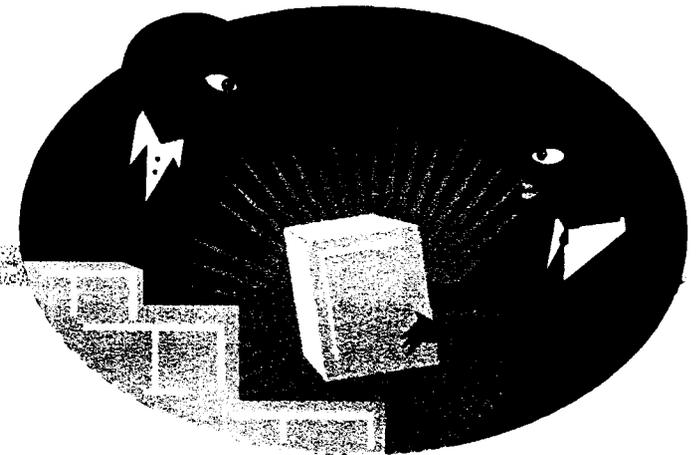
- Class Diagrams often used to show
  - The class hierarchy
  - Relationship between classes
  - Interfaces (public and protected methods only)
- Analyze if any
  - Design Patterns apply
  - Data Structures are needed

# High Level Design (cont'd)

---

- Object Diagram
- Sequence / Collaboration Diagram
- Activity Diagram
- State Diagram

# Detailed Design



- **All the diagrams**
- **Class' documentation includes**
  - Public, protected and private member
  - State charts
- **Operations**
  - English description with pseudo-code
  - Pre and post conditions
  - Assertions and testing criteria



# **Rational Rose**

---

## **CASE tool for OO Modeling**

**JPL**



# Overview



Document all stages of design

- Diagrams using UML notation

- Specification of classes, operations, attributes

- Rose Model is one stop for all documents

- Organize design in subsystems, and categories

- Highly customizable

# Generating Documents

---

## Documents we generate

- Interface Specification

- Class Interface
- File Interface

- Design Documents

- Create Visual Basic-like scripts to generate documents

- Rational Soda simplifies generating word documents using Rational Rose



# Code Generation and Reverse Engineering

---

- Forward Engineer

- Generates skeleton code for classes (both .h and .cpp files)

- Writes documentation in code

- Reverse Engineer existing code into the Model

- Keeping code and documentation in Sync

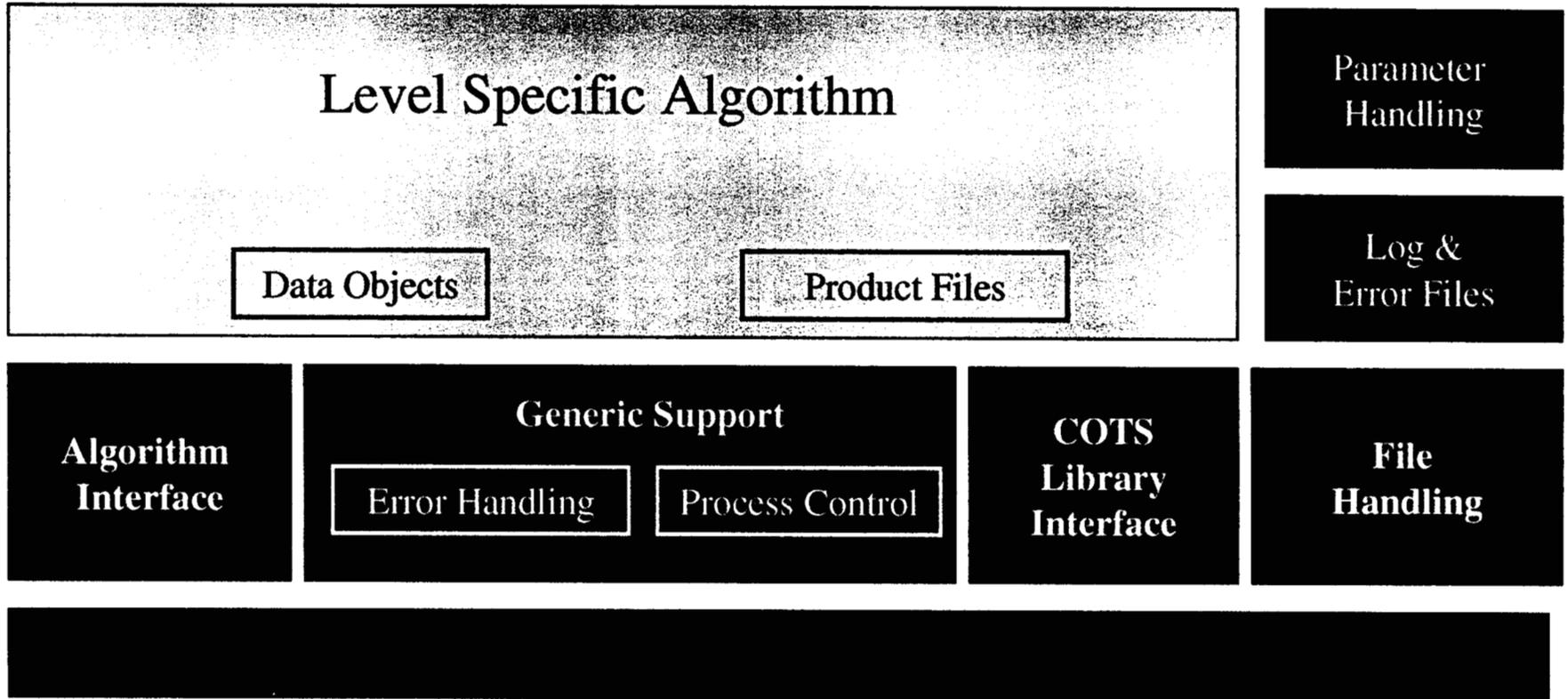


# Framework Design

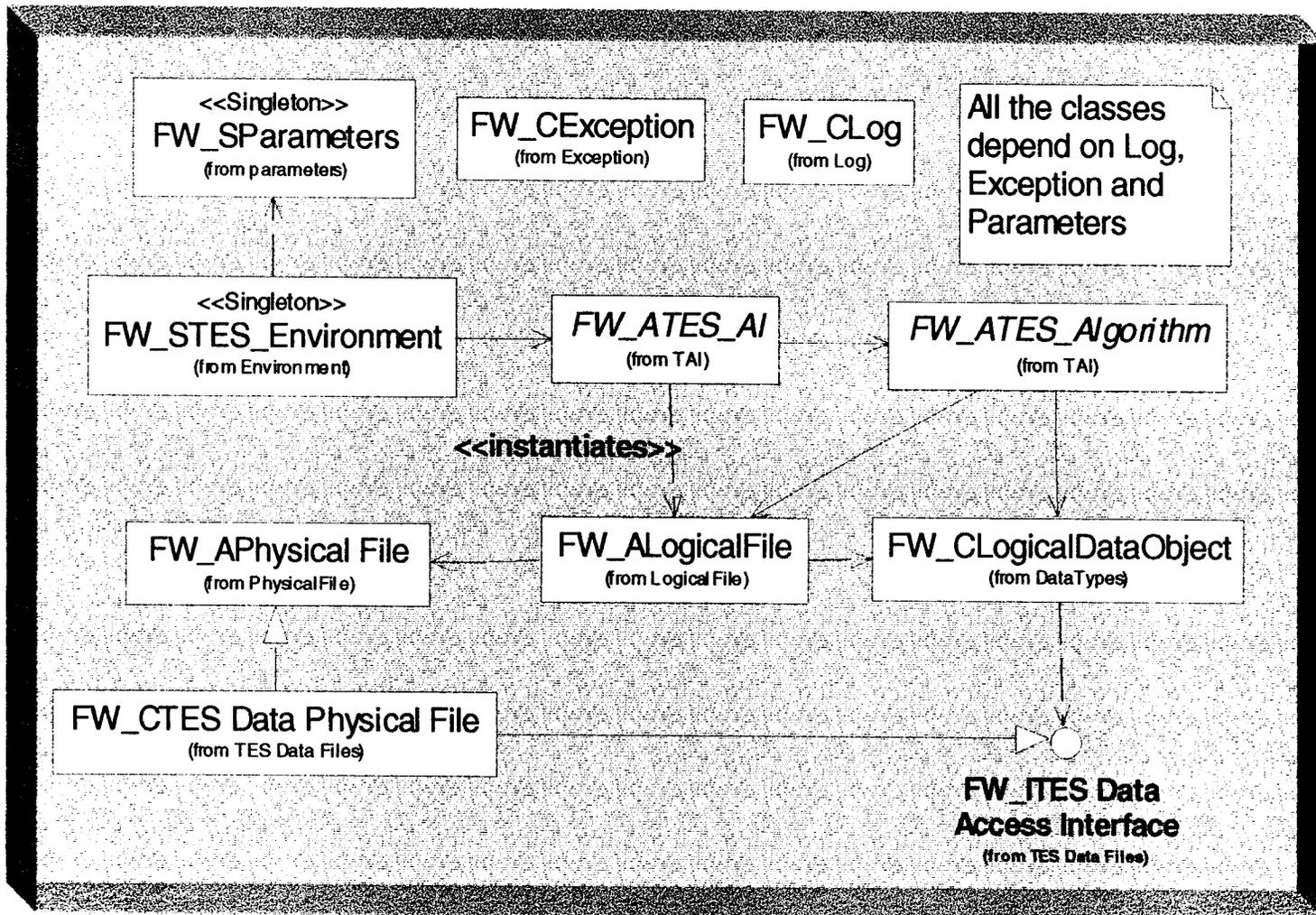
---



# SDPS Framework



# Design Overview

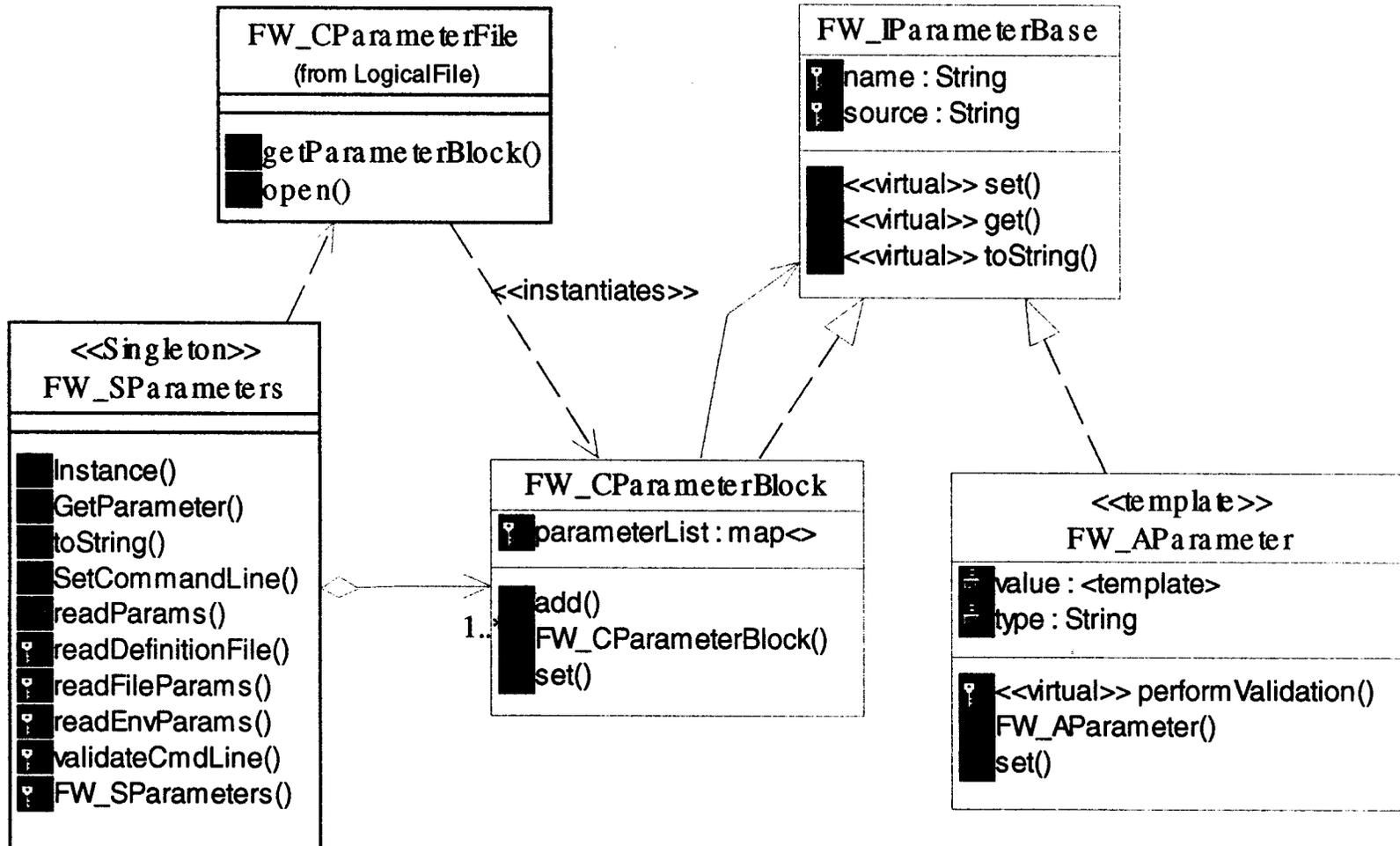


# Parameters



- Provides uniform access to all parameters from Command Line, Environment, and File Parameters
- Apply precedence rule to the parameters
- Global access provided using **Singleton** pattern
- The parameter file can be organized in nested blocks of parameters

# Parameter Hierarchy



# Framework Design

---

## Environment

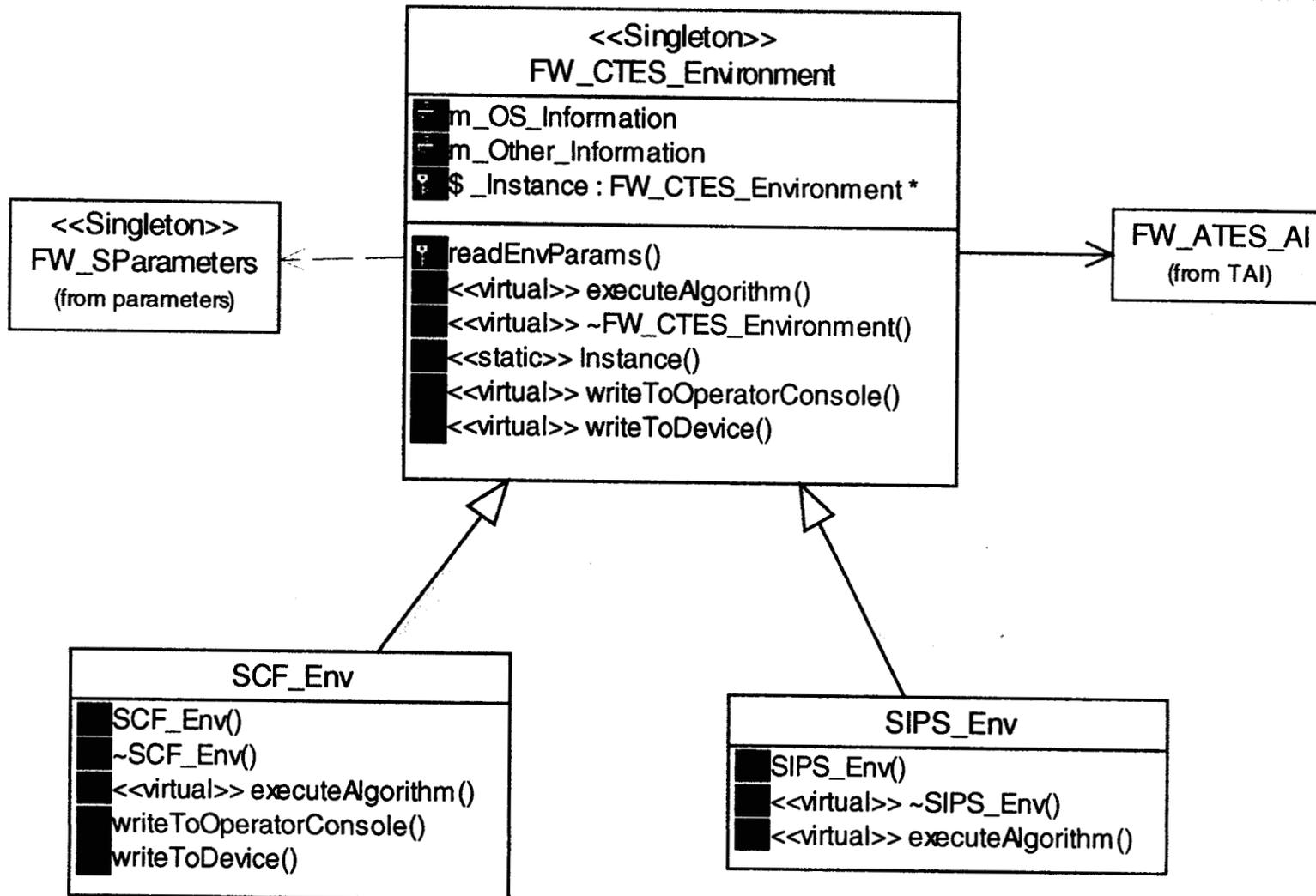


# Environment

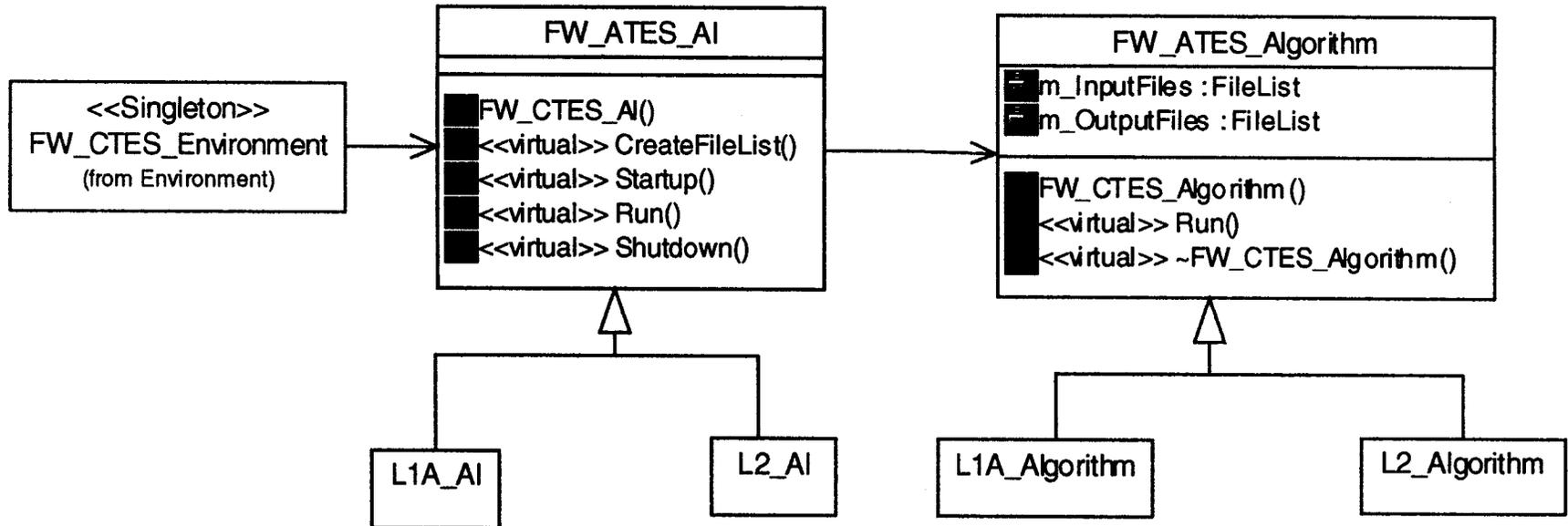
---

- Uniform interface for different environments
- Algorithms can run on different environments without any major changes
- Environment executes algorithms using Algorithm Interface (AI)
- AI is the middle man between Algorithm and Environment

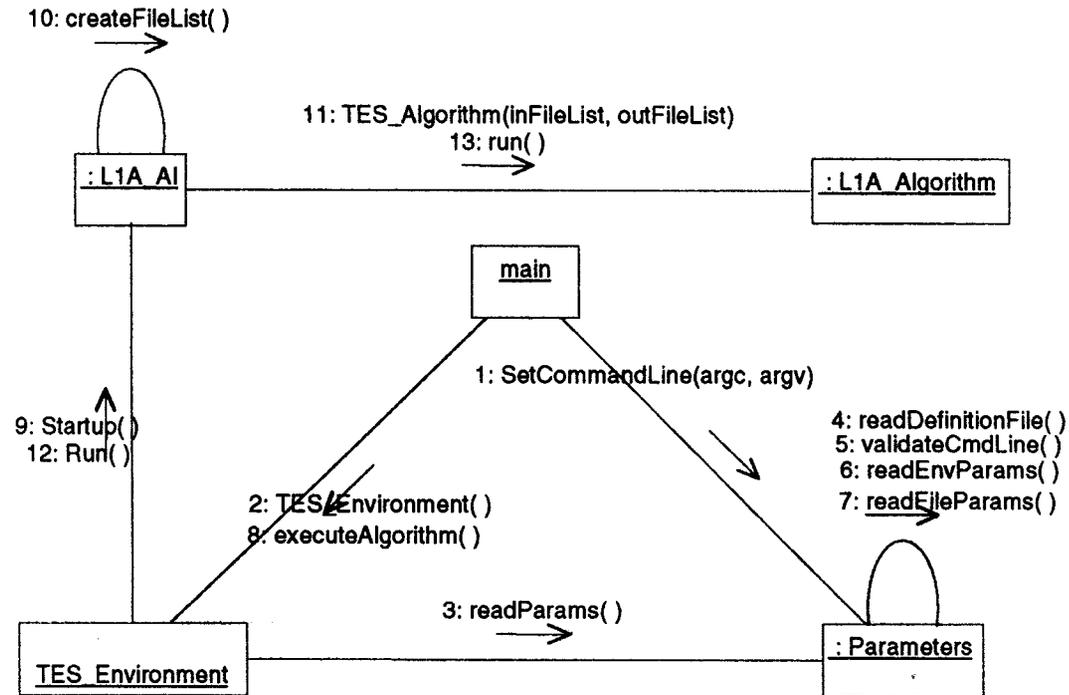
# Environment



# TES Algorithm Interface



# TES\_Environment Execution Diagram



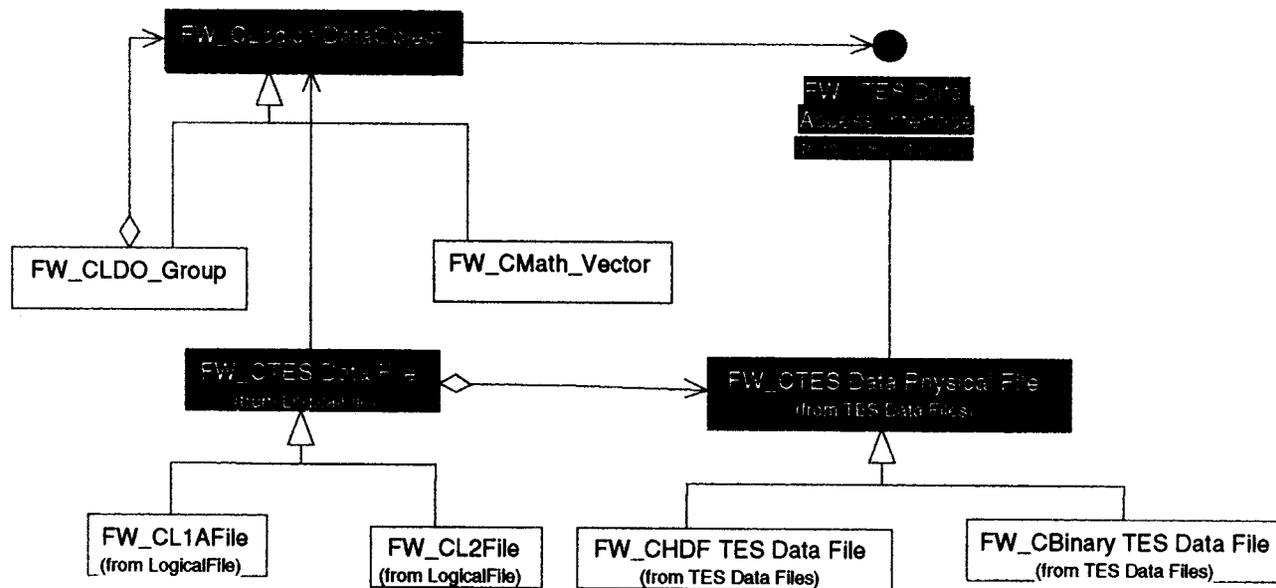
# Framework Design

---

## File I/O



# File I/O Overview

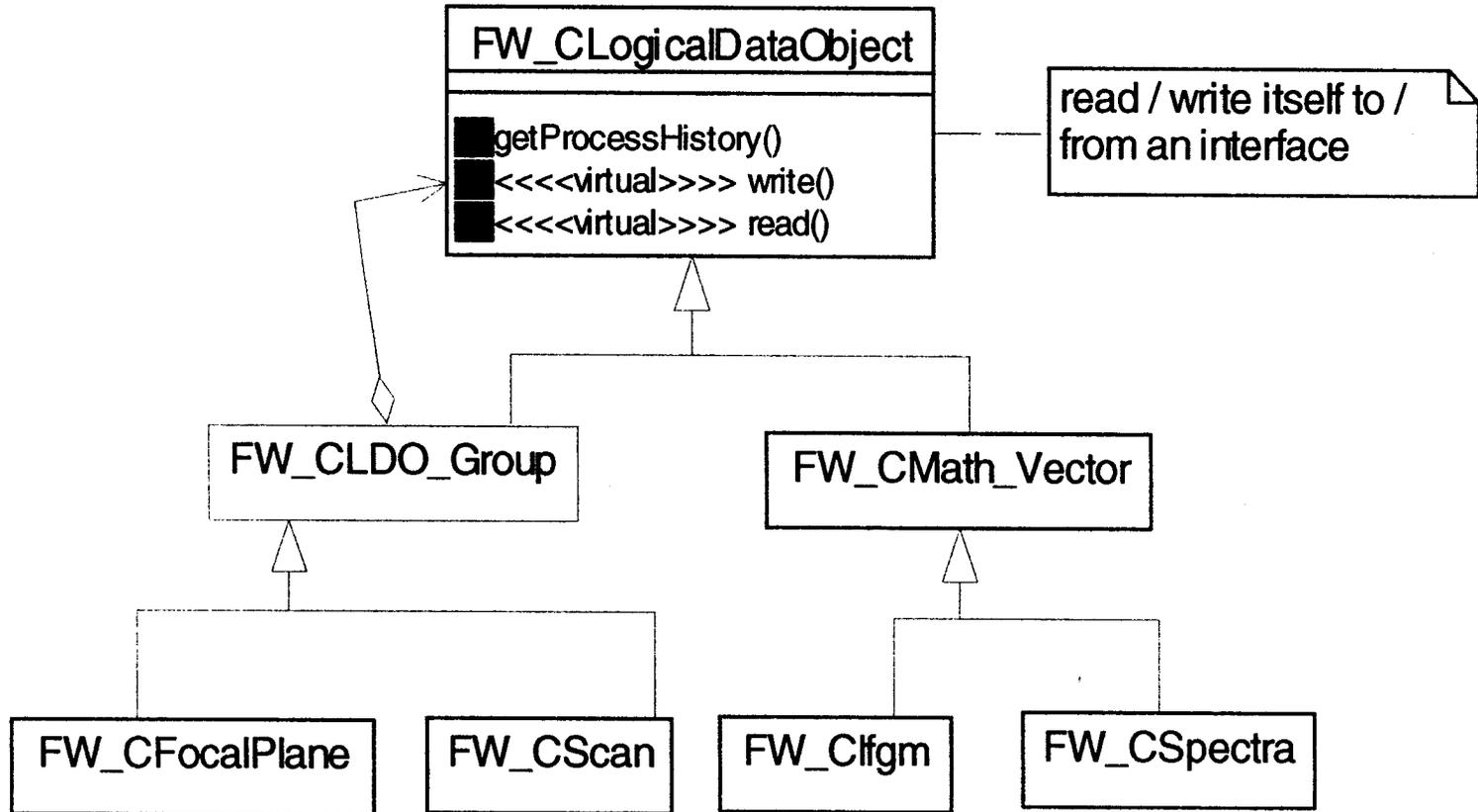


# Reading and Writing Data

---

- All TES Data Types should inherit from Logical Data Objects
- Logical Data Object, using the Data Access Interface, should read and write itself from a file
- Data Access Interface is overloaded by all TES Data Physical Files

# Logical Data Object



# Data Access Interface

```
<<Interface>>
FW_ITES Data Access Interface

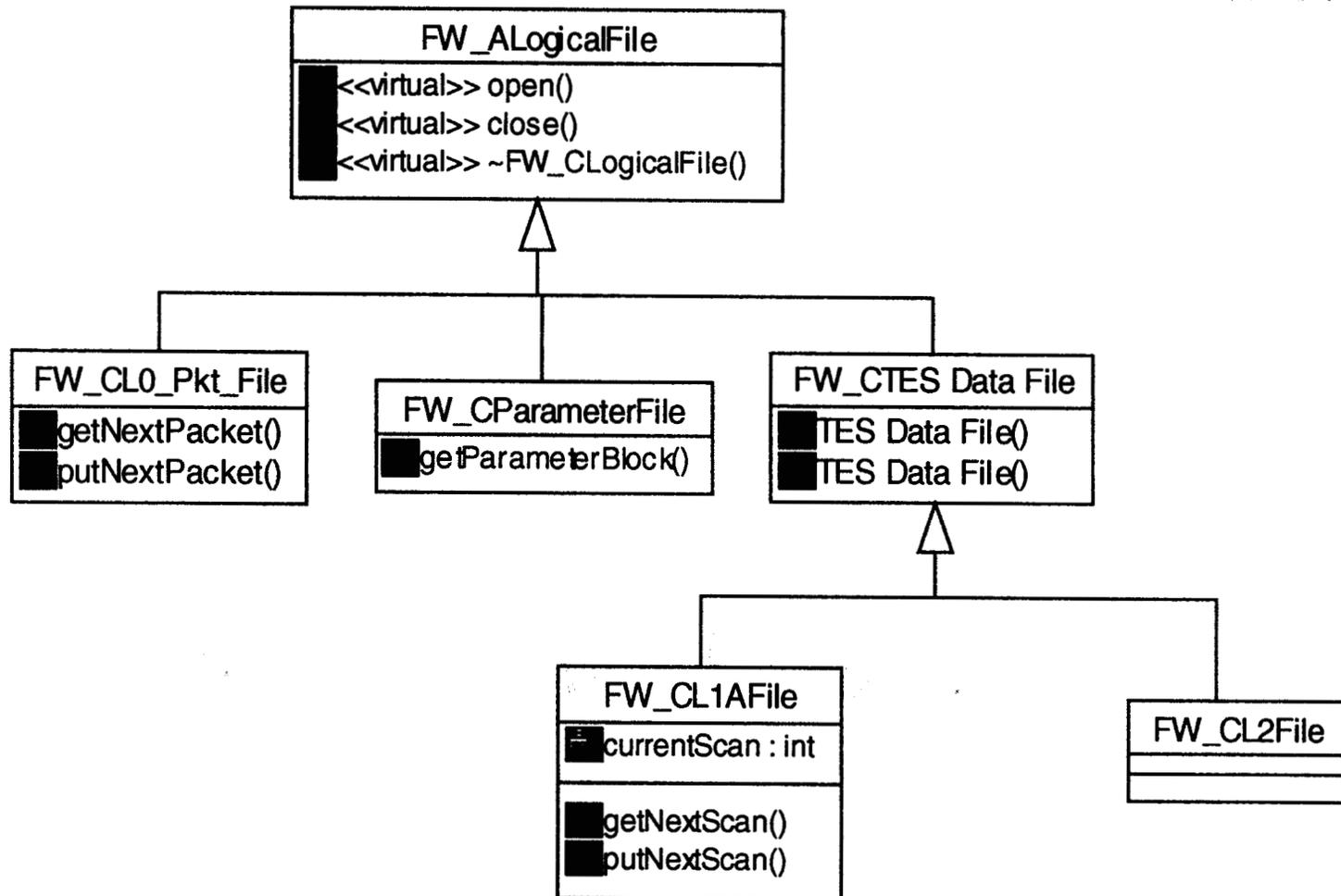
read(name, data : int_8&) : void
read(name, data : uint_8&) : void
read(name, data : int_16&) : void
read(name, data : uint_16&) : void
read(name, data : int_32&) : void
read(name, data : uint_32&) : void
read(name, data : float_32&) : void
read(name, data : float_64&) : void
read(name, data : char8&) : void
read(name, data : uchar8&) : void
read(name, data : String&) : void
read(name, data : BUFFER&) : void
openGroup(name : String, type : String) : void
closeGroup() : void
write(name, data : int_8) : void
write(name, data : uint_8) : void
write(name, data : int_16) : void
write(name, data : uint_16) : void
write(name, data : int_32) : void
write(name, data : uint_32) : void
write(name, data : float_32) : void
write(name, data : float_64) : void
write(name, data : char8) : void
write(name, data : uchar8) : void
write(name, data : BUFFER) : void
write(name, data : String) : void
```

# File I/O

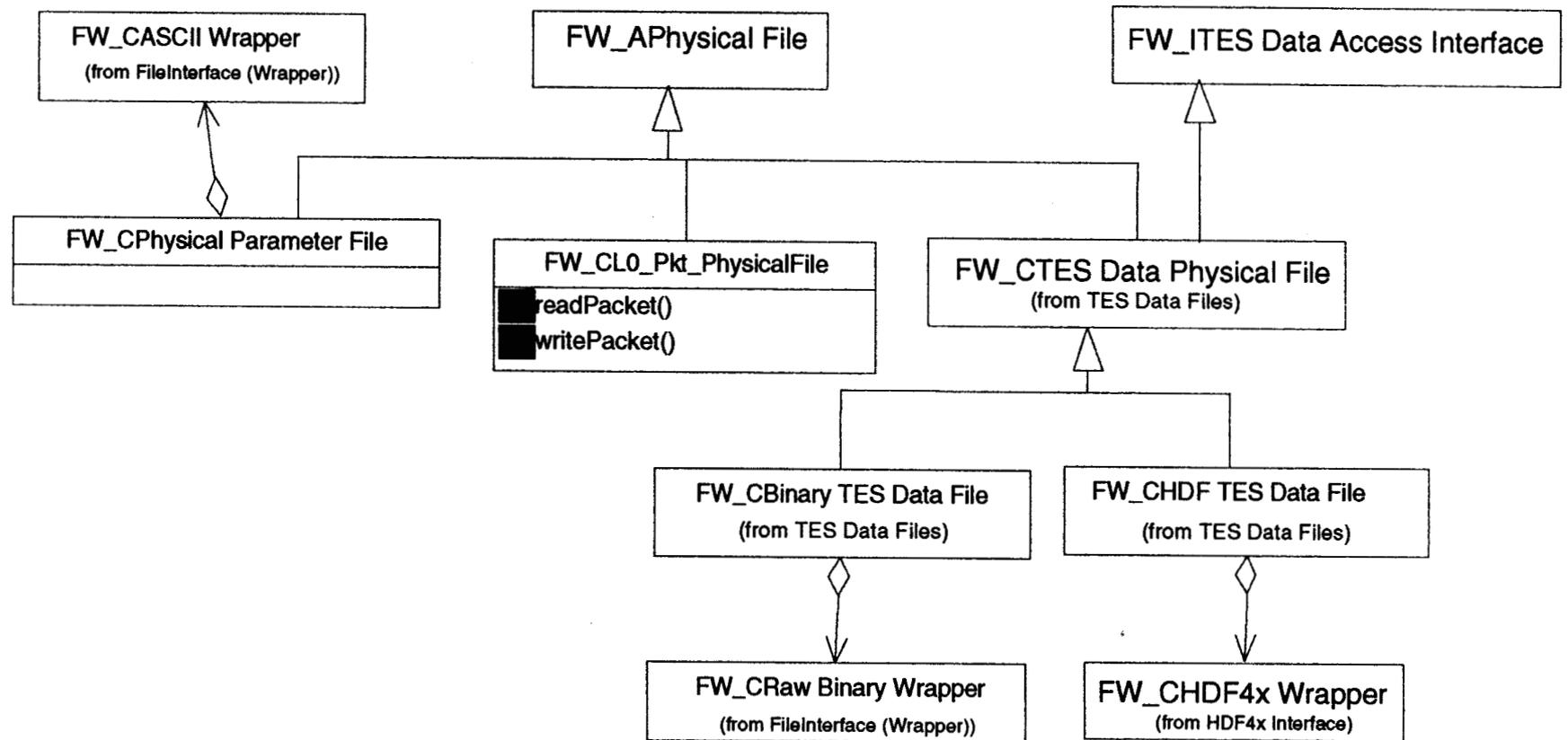


- Uses the **Bridge** pattern
  - Logical File provides an abstraction -- A class hierarchy for file interface
  - Physical File provides an implementation -- A class hierarchy for file type-specific implementation
- File Wrapper (**Adaptor** pattern) encapsulates the file API

# Logical File Diagram

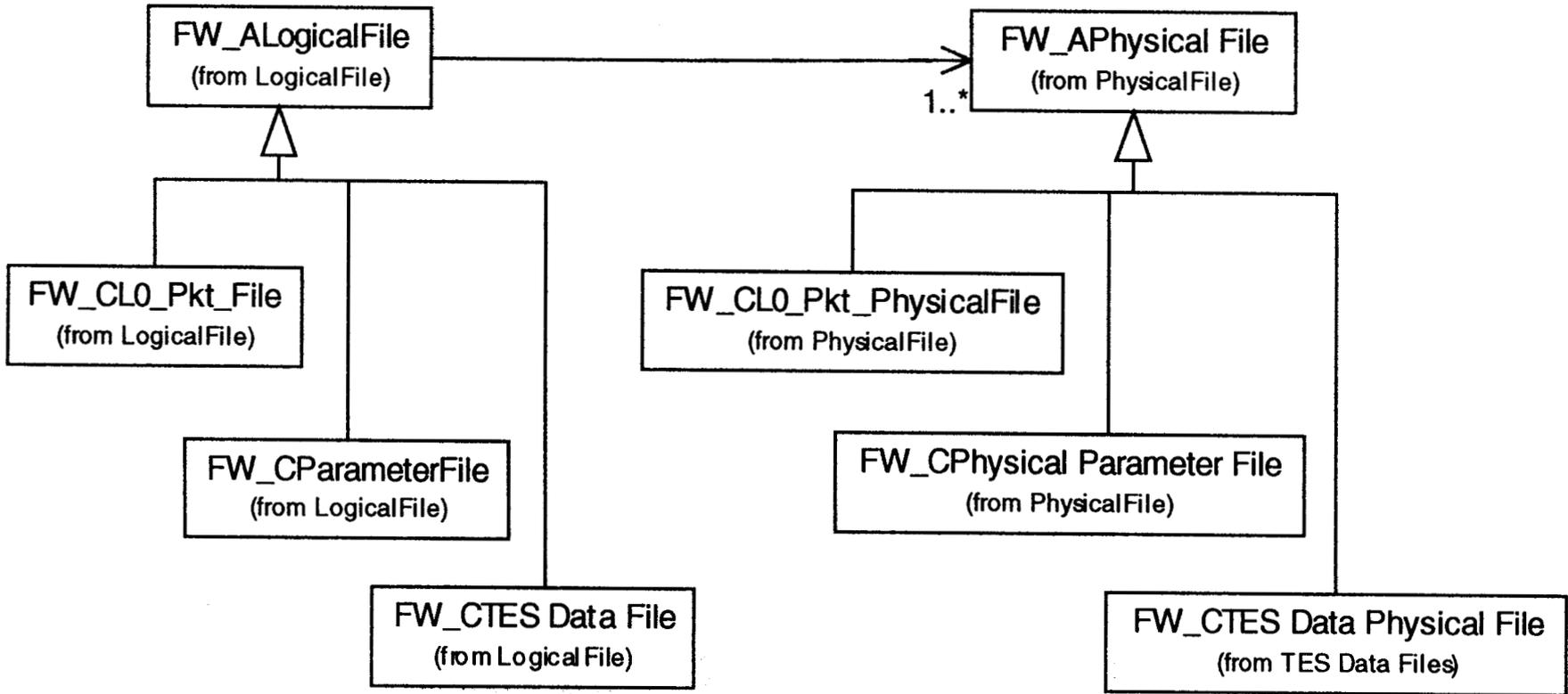


# Physical File Logical Diagram

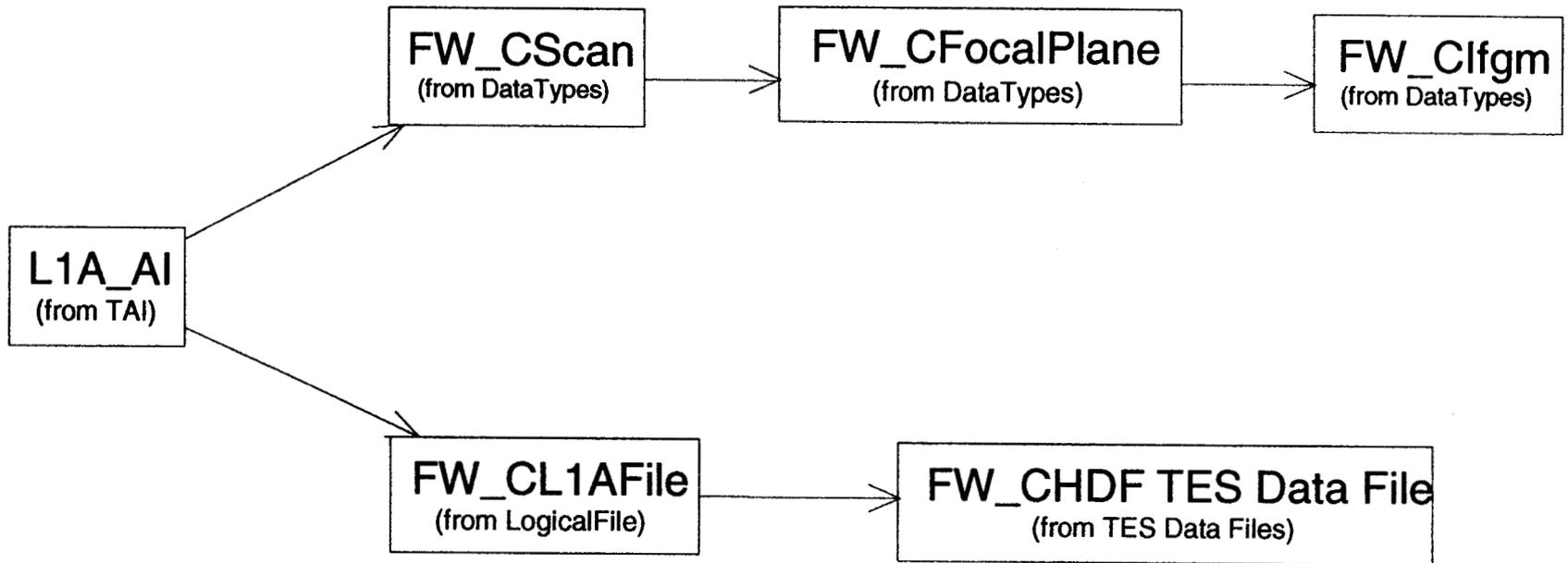


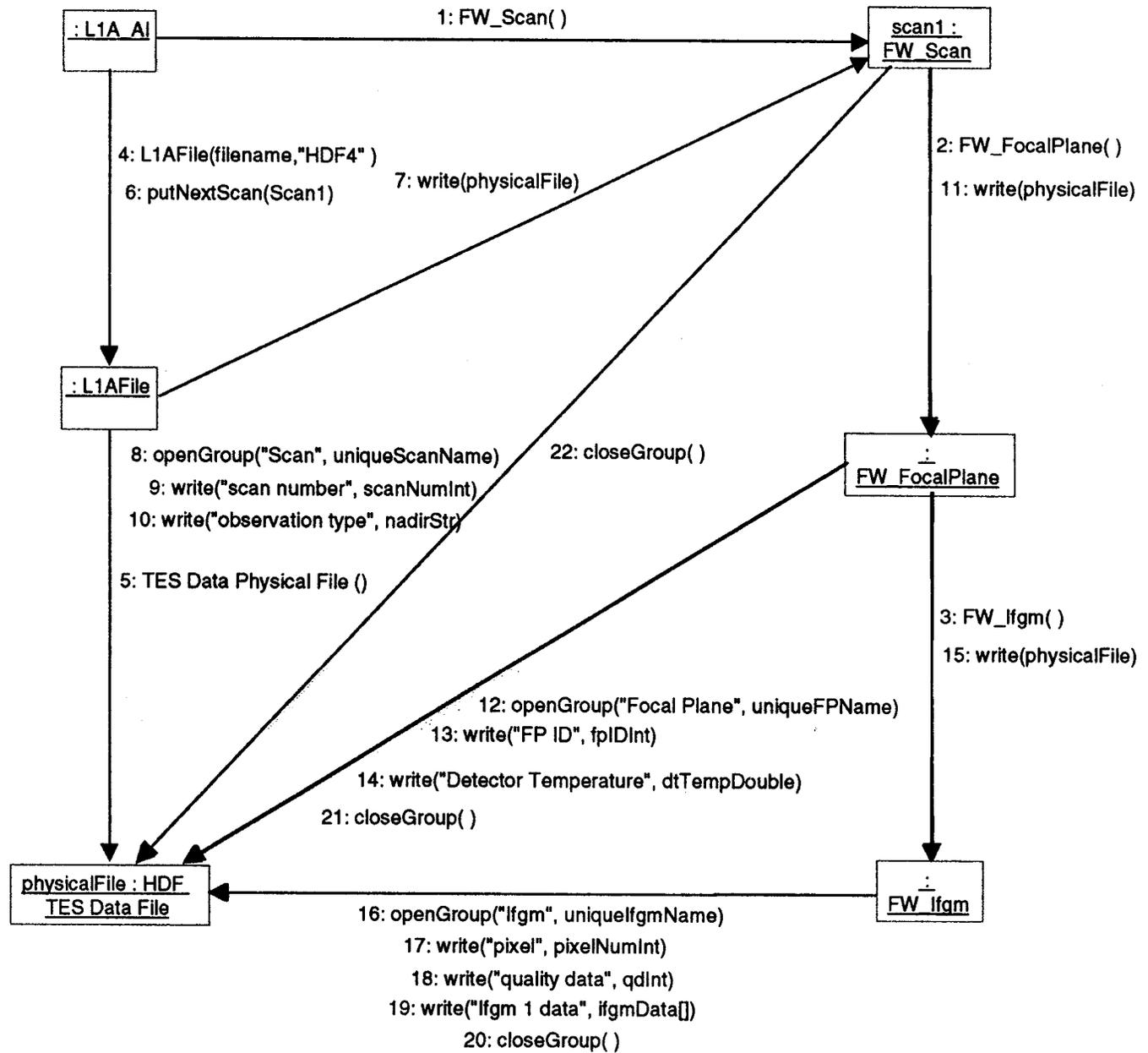
# Logical File Hierarchy

# Physical File Hierarchy



# File I/O Dependency



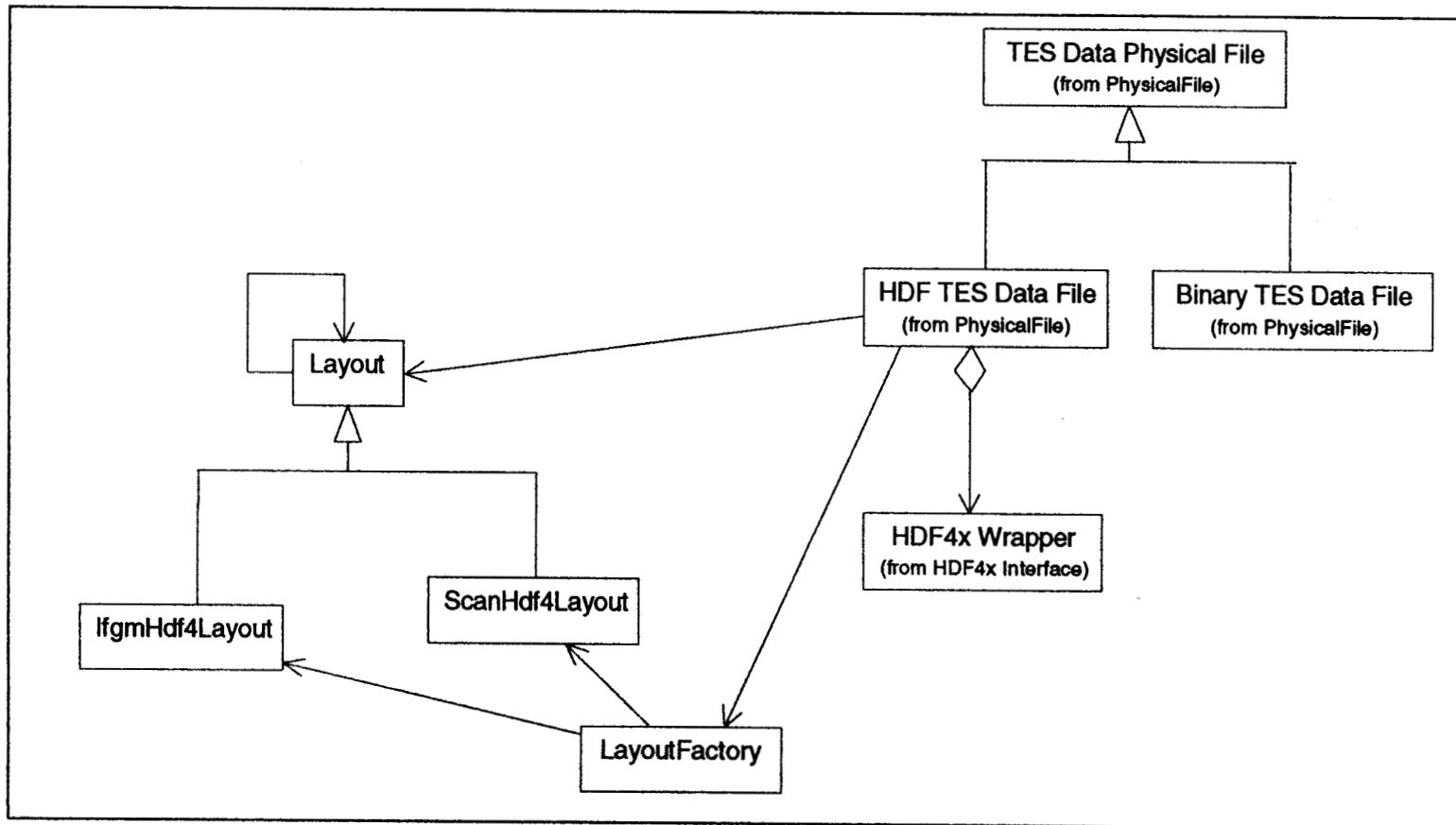


# File Layout Diagrams

---

- Every Logical Data Object (LDO) has an associated layout
- The Layout object specifies how to organize a LDO in a file (i.e. how the read/write calls are to be interpreted)
- Avoids coupling the Logical Data Objects with the File Layout by using the **Chain of Responsibility** pattern

# File-layout Diagram



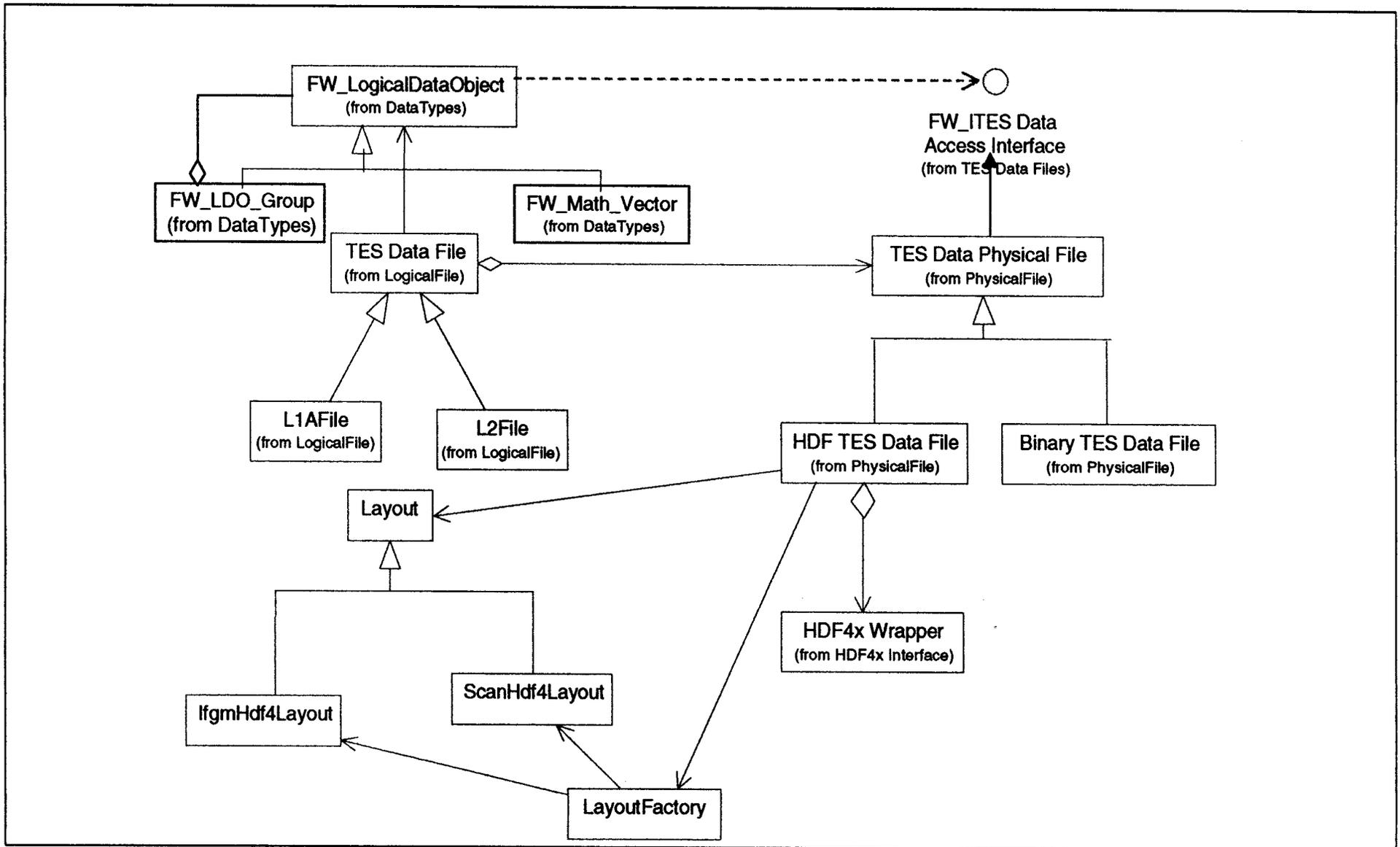
# Using Patterns

---

**JPL**



# Data Object-file Relationships

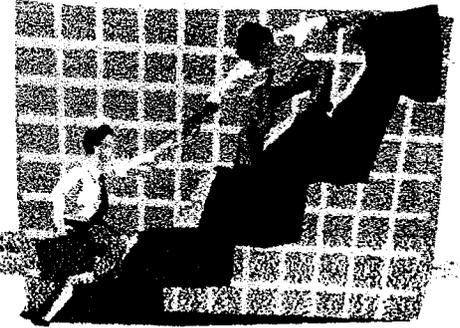


# Lessons Learned

---



# For Managers



- Selling the idea
- Framework should be a stand alone system
- Define scope of Framework and abide by it
- Iterative development fits best

# For Managers



- Design meetings and reviews benefit from small groups
- High learning curve for tools and methodology
- Training all team members is essential but expensive

# For Designers



- Communicating your design to new (or non) OO designers
  - Class diagrams can be too complicated
  - Object diagrams, Data Flow diagrams, and Block diagrams work best
- Traditional design tools are still very useful (e.g. DFD, State diagrams ...)

# Any Questions?

---



# Contacting the Author

---

Akbar Thobhani  
Software Engineer  
Jet Propulsion Laboratory  
4800 Oak Grove Drive  
Pasadena, CA 91109

[akbar.thobhani@jpl.nasa.gov](mailto:akbar.thobhani@jpl.nasa.gov)  
(818) 354-5158

# References

---

- Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, *Design Patterns*, Reading, MA: Addison-Wesley, 1995
- Steve Larson, Stephen Watson, Kalyani Rengarajan, *A Framework-Based Approach to Science Software Development, paper, IEEE 2000, document # 0-7803-5846-5/00*

# Acronyms

---

AI - Algorithm Interface

API - Application programming Interface

COTS - Commercial off-the-shelf software

FW - Framework

GOTS - Government off-the-shelf software

HDF - Hierarchical Data Format

JPL - Jet Propulsion Laboratory

L1A - Level 1A (subsystem of TES)

L2 - Level 2 (subsystem of TES)

LDO - Logical Data Object

NASA - National Aeronautics and Space Administration

OO - Object Oriented

PGE - Product Generation Executable

TAI - TES Algorithm Interface

TES - Tropospheric Emission Spectrometer

UML - Unified Modeling Language

# Acknowledgements

---

I would like to thank the following individuals for their valuable assistance

- Stephen Watson
- Ken Scott

The research described in this presentation was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration